# yaxin henglogo.jpg前言

None

## 作者

* 作者：戚辉永

# 目录

目录

[1. 前言 1](#_Toc373916828)

[1.1 作者 1](#_Toc373916829)

[2. 目录 1](#_Toc373916830)

[3. 阅读对象 1](#_Toc373916831)

[4. Zookeeper简介 2](#_Toc373916832)

[5. Zookeeper安装 2](#_Toc373916833)

[5.1 安装文件 2](#_Toc373916834)

[5.2 安装步骤 3](#_Toc373916835)

[6. Zookeeper示例 3](#_Toc373916836)

[6.1 示例代码 3](#_Toc373916837)

[6.2 常用API 5](#_Toc373916838)

[7. Zookeeper核心概念 6](#_Toc373916839)

[7.1 Znodes 6](#_Toc373916840)

[8. 互联网资料 7](#_Toc373916841)

[9. 参考 7](#_Toc373916842)

# 阅读对象

本文档使用对分布式开发有初步认识的开发人员，如果不知道分布式开发的概念请上网参考相关资料，这方面的资料非常多。

# Zookeeper简介

Zookeeper是一个高性能，分布式的，开源分布式应用协调服务，它用于解决分布式系统之间的协调问题，如：分布式锁、同步队列、集群管理等。

**ZooKeeper 典型的应用场景**

Zookeeper 从设计模式角度来看，是一个基于观察者模式设计的分布式服务管理框架，它负责存储和管理大家都关心的数据，然后接受观察者的注册，一旦这些数据的状态发生变化，Zookeeper 就将负责通知已经在 Zookeeper 上注册的那些观察者做出相应的反应，从而实现集群中类似 Master/Slave 管理模式，关于 Zookeeper 的详细架构等内部细节可以阅读 Zookeeper 的源码

下面详细介绍这些典型的应用场景，也就是 Zookeeper 到底能帮我们解决那些问题？下面将给出答案。

**统一命名服务（Name Service）**

分布式应用中，通常需要有一套完整的命名规则，既能够产生唯一的名称又便于人识别和记住，通常情况下用树形的名称结构是一个理想的选择，树形的名称结构是一个有层次的目录结构，既对人友好又不会重复。说到这里你可能想到了 JNDI，没错 Zookeeper 的 Name Service 与 JNDI 能够完成的功能是差不多的，它们都是将有层次的目录结构关联到一定资源上，但是 Zookeeper 的 Name Service 更加是广泛意义上的关联，也许你并不需要将名称关联到特定资源上，你可能只需要一个不会重复名称，就像数据库中产生一个唯一的数字主键一样。

Name Service 已经是 Zookeeper 内置的功能，你只要调用 Zookeeper 的 API 就能实现。如调用 create 接口就可以很容易创建一个目录节点。

**配置管理（Configuration Management）**

配置的管理在分布式应用环境中很常见，例如同一个应用系统需要多台 PC Server 运行，但是它们运行的应用系统的某些配置项是相同的，如果要修改这些相同的配置项，那么就必须同时修改每台运行这个应用系统的 PC Server，这样非常麻烦而且容易出错。

像这样的配置信息完全可以交给 Zookeeper 来管理，将配置信息保存在 Zookeeper 的某个目录节点中，然后将所有需要修改的应用机器监控配置信息的状态，一旦配置信息发生变化，每台应用机器就会收到 Zookeeper 的通知，然后从 Zookeeper 获取新的配置信息应用到系统中。

# Zookeeper安装

## 安装文件

* zookeeper-3.4.5.tar.gz最新的安装包

下载地址：

<http://mirror.bit.edu.cn/apache/zookeeper/zookeeper-3.4.5/zookeeper-3.4.5.tar.gz>

## 安装步骤

1. 使用 tar -xzvf zookeeper-3.4.2.tar.gz  解压
2. 设置，将conf/zoo.example.cfg复制到conf/zoo.cfg或者手动建立一个新的。
3. 启动Zookeeper服务：bin/zkServer.sh start
4. 启动客户端连接：bin/zkCli.sh -server 127.0.0.1:2181（此处在本机，且使用了默认端口，且在Java环境中）
5. yaxin henglogo.jpg关闭Zookeeper服务：bin/zkServer.sh stop

# Zookeeper示例

## 示例代码

* Helloword代码：

**public** **static** **void** main(String[] args) **throws** Exception {

ZooKeeper zk =**new** ZooKeeper("10.1.27.36:2181", 500000, **new** Watcher() {

**public** **void** process(WatchedEvent event) {

System.*out*.println("1触发了 "+event.getType()+" 事件！");

}

});

// 创建一个目录节点

System.*out*.println("操作增加/testRootPath节点 ");

zk.exists("/testRootPath/testChildPathOne",**true**);

zk.create("/testRootPath", "testRootData".getBytes(), Ids.*OPEN\_ACL\_UNSAFE*,

CreateMode.*PERSISTENT*);

// 创建一个子目录节点

System.*out*.println("操作增加/testRootPath/testChildPathOne子节点 ");

zk.create("/testRootPath/testChildPathOne", "testChildDataOne".getBytes(),

Ids.*OPEN\_ACL\_UNSAFE*,CreateMode.*PERSISTENT*);

System.*out*.println("操作获取/testRootPath节点 数据，=========》 ");

System.*out*.println(**new** String(zk.getData("/testRootPath",**false**,**null**)));

// 取出子目录节点列表

System.*out*.println("操作获取/testRootPath子节点，=========》 ");

System.*out*.println(zk.getChildren("/testRootPath",**true**));

// 修改子目录节点数据

System.*out*.println("操作修改/testRootPath/testChildPathOne节点 数据，=========》 ");

zk.setData("/testRootPath/testChildPathOne","modifyChildDataOne".getBytes(),-1);

System.*out*.println("目录节点状态：["+zk.exists("/testRootPath",**true**)+"]");

// 创建另外一个子目录节点

System.*out*.println("操作增加/testRootPath/testChildPathTwo子节点 ");

zk.create("/testRootPath/testChildPathTwo", "testChildDataTwo".getBytes(),

Ids.*OPEN\_ACL\_UNSAFE*,CreateMode.*PERSISTENT*);

System.*out*.println("操作获取/testRootPath/testChildPathTwo节点 数据，=========》 ");

System.*out*.println(**new** String(zk.getData("/testRootPath/testChildPathTwo",**true**,**null**)));

// 删除子目录节点

zk.exists("/testRootPath/testChildPathOne",**true**);

System.*out*.println("删除/testRootPath/testChildPathTwo节点 ");

zk.delete("/testRootPath/testChildPathTwo",-1);

System.*out*.println("删除/testRootPath/testChildPathOne节点 ");

zk.delete("/testRootPath/testChildPathOne",-1);

// 删除父目录节点

System.*out*.println("删除/testRootPath节点 ");

zk.delete("/testRootPath",-1);

// 关闭连接

zk.close();

Thread.*sleep*(100000);

}

* 控制台输出

操作增加/testRootPath节点

操作增加/testRootPath/testChildPathOne子节点

操作获取/testRootPath节点 数据，=========》

testRootData

操作获取/testRootPath子节点，=========》

[testChildPathOne]

操作修改/testRootPath/testChildPathOne节点 数据，=========》

目录节点状态：[229,229,1386121687735,1386121687735,0,1,0,0,12,1,230

]

操作增加/testRootPath/testChildPathTwo子节点

操作获取/testRootPath/testChildPathTwo节点 数据，=========》

testChildDataTwo

删除/testRootPath/testChildPathTwo节点

删除/testRootPath/testChildPathOne节点

删除/testRootPath节点

1触发了 None 事件！

1触发了 NodeCreated 事件！

1触发了 NodeChildrenChanged 事件！

1触发了 NodeDeleted 事件！

1触发了 NodeDeleted 事件！

1触发了 NodeDeleted 事件！

## 常用API

|  |  |
| --- | --- |
| **方法名** | **方法功能描述** |
| [String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) [create](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#create%28java.lang.String,%20byte[],%20java.util.List,%20org.apache.zookeeper.CreateMode%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, byte[] data, [List](http://java.sun.com/javase/6/docs/api/java/util/List.html?is-external=true)<[ACL](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/ACL.html)> acl,[CreateMode](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/CreateMode.html) createMode) | 创建一个给定的目录节点 path, 并给它设置数据，[CreateMode](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/CreateMode.html) 标识有四种形式的目录节点，分别是 PERSISTENT：持久化目录节点，这个目录节点存储的数据不会丢失；PERSISTENT\_SEQUENTIAL：顺序自动编号的目录节点，这种目录节点会根据当前已近存在的节点数自动加 1，然后返回给客户端已经成功创建的目录节点名；EPHEMERAL：临时目录节点，一旦创建这个节点的客户端与服务器端口也就是 session 超时，这种节点会被自动删除；EPHEMERAL\_SEQUENTIAL：临时自动编号节点 |
| [Stat](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/Stat.html) [exists](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#exists%28java.lang.String,%20boolean%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, boolean watch) | 判断某个 path 是否存在，并设置是否监控这个目录节点，这里的 watcher 是在创建 ZooKeeper 实例时指定的 watcher，[exists](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html" \l "exists%28java.lang.String,%20boolean%29)方法还有一个重载方法，可以指定特定的watcher |
| [Stat](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/Stat.html) [exists](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#exists%28java.lang.String,%20org.apache.zookeeper.Watcher%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path,[Watcher](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/Watcher.html) watcher) | 重载方法，这里给某个目录节点设置特定的 watcher，Watcher 在 ZooKeeper 是一个核心功能，Watcher 可以监控目录节点的数据变化以及子目录的变化，一旦这些状态发生变化，服务器就会通知所有设置在这个目录节点上的 Watcher，从而每个客户端都很快知道它所关注的目录节点的状态发生变化，而做出相应的反应 |
| void [delete](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#delete%28java.lang.String,%20int%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, int version) | 删除 path 对应的目录节点，version 为 -1 可以匹配任何版本，也就删除了这个目录节点所有数据 |
| [List](http://java.sun.com/javase/6/docs/api/java/util/List.html?is-external=true)<[String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true)>[getChildren](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html" \l "getChildren%28java.lang.String,%20boolean%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, boolean watch) | 获取指定 path 下的所有子目录节点，同样 [getChildren](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html" \l "getChildren%28java.lang.String,%20boolean%29)方法也有一个重载方法可以设置特定的 watcher 监控子节点的状态 |
| [Stat](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/Stat.html) [setData](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html" \l "setData%28java.lang.String,%20byte[],%20int%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, byte[] data, int version) | 给 path 设置数据，可以指定这个数据的版本号，如果 version 为 -1 怎可以匹配任何版本 |
| byte[] [getData](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html" \l "getData%28java.lang.String,%20boolean,%20org.apache.zookeeper.data.Stat%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, boolean watch, [Stat](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/Stat.html) stat) | 获取这个 path 对应的目录节点存储的数据，数据的版本等信息可以通过 stat 来指定，同时还可以设置是否监控这个目录节点数据的状态 |
| void[addAuthInfo](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#addAuthInfo%28java.lang.String,%20byte[]%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) scheme, byte[] auth) | 客户端将自己的授权信息提交给服务器，服务器将根据这个授权信息验证客户端的访问权限。 |
| [Stat](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/Stat.html) [setACL](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html" \l "setACL%28java.lang.String,%20java.util.List,%20int%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path,[List](http://java.sun.com/javase/6/docs/api/java/util/List.html?is-external=true)<[ACL](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/ACL.html)> acl, int version) | 给某个目录节点重新设置访问权限，需要注意的是 Zookeeper 中的目录节点权限不具有传递性，父目录节点的权限不能传递给子目录节点。目录节点 ACL 由两部分组成：perms 和 id。 Perms 有 ALL、READ、WRITE、CREATE、DELETE、ADMIN 几种  而 id 标识了访问目录节点的身份列表，默认情况下有以下两种： ANYONE\_ID\_UNSAFE = new Id("world", "anyone") 和 AUTH\_IDS = new Id("auth", "") 分别表示任何人都可以访问和创建者拥有访问权限。 |
| [List](http://java.sun.com/javase/6/docs/api/java/util/List.html?is-external=true)<[ACL](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/ACL.html)>[getACL](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html" \l "getACL%28java.lang.String,%20org.apache.zookeeper.data.Stat%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path,[Stat](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/Stat.html) stat) | 获取某个目录节点的访问权限列表 |

# Zookeeper核心概念

## Znodes

Zookeeper管理着一个树形结构，在Zookeeper树种每一个节点称为znode。Zondes维持一状态结构包含了数据改变版本号和ACL（Access Control List）改变的版本号。这个静态结构也包含了时间戳。这些版本号和时间戳在一起用于zookeeper验证缓存和协调更新。

当znodes数据改变后这个版本后会增加。如果客户端检索znode的数据也会检索这个版本号。并且当客户端执行一个更新、删除，它也必须提供znode版本数据，如果提供的版本号与数据的版本号不匹配更新将失败。

**Note**

In distributed application engineering, the word node can refer to a generic host machine, a server, a member of an ensemble, a client process, etc. In the ZooKeeper documentation, znodes refer to the data nodes. Servers refer to machines that make up the ZooKeeper service; quorum peers refer to the servers that make up an ensemble; client refers to any host or process which uses a ZooKeeper service.

Znodes是主要的程序访问实体、他们有几个特征：

* **观察者**

客户端可以设置zonodes观察者。改变了znode它就会触发然后清楚观察者。当观察者被触发，zookeeper给客户端发送一个通知。更多关于Watches信息请参考[ZooKeeper Watches](http://zookeeper.apache.org/doc/r3.4.5/zookeeperProgrammers.html" \l "ch_zkWatches)。

* **数据访问**

数据存储在每一个znode在一个命名空间读和写是原子的。读获得关于这个znode所有的数据字节，和写替换所有的数据。每一个node有一个访问控制列表（ACL）这里约束了谁能做什么。（以下是官方原文）。

The data stored at each znode in a namespace is read and written atomically. Reads get all the data bytes associated with a znode and a write replaces all the data. Each node has an Access Control List (ACL) that restricts who can do what.

Zookeeper设计不是用于通用数据库或大对象的存储。相反他管理者协调数据。这些数据可以是一些配置数据、状态数据、coordination数据等等。这些协调的公共数据相对是比较小的千字节以内。Zookeeper客户端和服务端的健康检查标准是这些znodes不超过1M数据，但是实际平均数据应该比这个小的多。操作相对较大的数据就会耗掉很多的时间，这样其他系统会形成阻塞。如果确实需要操作大的数据一般znode存放数据的位置，实际数据放在HDFS上。

* **临时Nodes**

Zookeeper也包含一个临时节点的概念。这些znode存活时间和session的创建一样长。当这个session结束znode就会删除。因为这种行为临时znode不允许包含children。

* **Sequence Nodes**

当创建一个znode你也可以要求zookeeper最佳一个递增的计数器在路径尾端。这个计数器在上级node中是唯一的。这个计数格式化为%010d。

# 互联网资料

* **Zookeeper浏览器(方便查看管理zookeeper)**

<https://github.com/killme2008/node-zk-browser>

* **ZooKeeper之特性**

ZooKeeper本质上是一个分布式的小文件存储系统。原本是Apache Hadoop的一个组件，现在被拆分为一个Hadoop的独立子项目，在HBase（Hadoop的另外一个被拆分出来的子项目，用于分布式环境下的超大数据量的DBMS）中也用到了ZooKeeper集群。ZooKeeper有如下的特性：

1）   简单

ZooKeeper核心是一个精简的文件系统，它提供了一些简单的文件操作以及附加的功能，例如排序和通知。

2）   易表达

ZooKeeper的数据结构原型是一棵znode树（类似Linux的文件系统），并且它们是一些已经被构建好的块，可以用来构建大型的协作数据结构和协议。

3）   高可用性

ZooKeeper可以运行在一组服务器上，同时它们被设计成高可用性，为你的应用程序避免单点故障。

4）   松耦合交互

ZooKeeper提供的Watcher机制使得各客户端与服务器的交互变得松耦合，每个客户端无需知晓其他客户端的存在，就可以和其他客户端进行数据交互。

5）   丰富的API

ZooKeeper为开发人员提供了一套丰富的API，减轻了开发人员编写通用协议的负担。

# 参考

官方资料：<http://zookeeper.apache.org/doc/r3.4.5/zookeeperProgrammers.html>

**IBM网站：**[**http://www.ibm.com/developerworks/cn/opensource/os-cn-zookeeper/**](http://www.ibm.com/developerworks/cn/opensource/os-cn-zookeeper/)